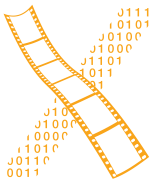

Chair for Computer Science 10 (Media Computing and Human-Computer Interaction)

RWTH AACHEN UNIVERSITY

Embroidered touchpad sensors

Thesis
submitted to the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Philip Kindermann

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr. Bastian Leibe

Registration date: 16.01.2018
Submission date: 24.04.2018

Eidesstattliche Versicherung

_____ _____

Name, Vorname Matrikelnummer

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel

_____

_____

_____

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

_____ _____

Ort, Datum Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

_____ _____

Ort, Datum Unterschrift

Contents

List of Figures

Abstract

Textile touch pad sensors are fairly developed in scientific literature and are even reaching into the consumer market. Even though this is the case textile touch pads are still inaccessible for most even in the do-it-your-self (DIY) community. They require lots of labor to create and connect multiple conductive and non-conductive layers in such a manner that they can be used as a textile touch pad. With this bachelor thesis we create an easily accessible textile touch pad with the use of an embroidery machine.
Additionally we want this design to be at best on a single layer so that is easily integrated into existing fabric e.g. clothing, furniture and the like. Given our tools someone should be able to recreate our sensor design and use it for his creative process. The tools we use are easily accessible embroidery machines in places like fablabs or public institutions as well as conductive thread that can be found at many retailers.

Überblick

Textile Touch-Pad-Sensoren sind ziemlich fortgeschritten in der wissenschaftlicher Literatur und reichen sogar in den Konsumentenmarkt. Doch obwohl dass der Fall ist, sind textile Touch-Sensoren nicht erwerblich für die meisten selbst für Leute die aktiv in do-it-yourself (DIY) Gemeinschaften sind. Aktuelle Ansätze erfordern viel Arbeit und verbinden mehrere leitfähige und nicht leitfähige Lagen mit einander sodass diese zusammen als textiler Touch-Sensor genutzt werden kann. Mit dieser Bachelorarbeit entwickeln wir einen leicht herzustellenden Sensor, der mit einer Stickmaschine hergestellt werden kann.
Zusätzlich wollen wir, dass dieses Design im besten Fall aus einer einzigen Lage besteht und so leicht in bestehende Stoffe und Kleidungsstücke integriert werden kann. Jemand der unsere Werkzeuge gegeben hat sollte in der Lage sein unseren Sensor zu rekreieren und für seine kreativen Zwecke zu nutzen. Die Werkzeuge die wie dafür benutzen sind leicht zugängliche Stickmaschinen die man in FabLabs oder öffentlichen Einrichtungen vortreffen kann. Der zugehörige leitfähige Faden ist erhältlich bei vielen Einzelhändlern zu günstigen Preisen.

Acknowledgements

I would like to thank Prof. Dr. Jan Borchers for supervising my thesis and Prof. Dr. Bastian Leibe for being my second examiner.
Furthermore I want to thank Nur Al-huda Hamdan for her feedback and guidance. Finally I want to thank Jan Thar, Christian Schmidt and all members of the Media Computing Group at RWTH that provided me with feedback.

Conventions

The whole thesis is written in American English.
This thesis is written in the first person plural due to aesthetic reasons.

Chapter 1

Introduction

DIY textile touchpads

Each year the maker community world wide is growing steadily and with it the availability to their tools to enhance their possessions with smart textiles. Anyone can go to their local Fab Lab, library or public institutions to gain access to an embroidery machine. Combined with widespread conductive yarns that are easily accessible to buy online they can craft their own textile sensors to use for anything they can think of. To ease the creation of these sensors allows makers of all skill levels to think of new ways to implement and improve them to further the field and benefit it as a whole.

Smart textiles on consumer level

Google's Project Jacquard by Ivan Poupyrev and Robinson [2016] collaborated with Levi to created the first consumer product implementing conductive yarn into clothing on a large scale. Commuter x Google Jacquard the smart denim jacket created within this collaboration is able to recognize simple gestures on it's sleeve to control your smartphone in whatever way you program those gestures to. This brought smart clothing into the public light and created interest as well as criticism. The commuter jacket shows the limits of smart clothing on a large scale that also exist on smaller scales in self made prototypes.

This thesis focuses on easily created and integrated textile touch pads that can be used broadly in every day applications. That may include clothing, furniture to standalone fabric controllers. Textile touch pads can be more intuitive to use in a hands free scenario guiding the interaction with

their physical properties. Furthermore they can provide a friendlier, warmer user experience.

Capacitive and resistive touch

Touch interfaces nowadays are mainly capacitive touch screens in smartphones while there reside multiple way to realize touch input. Opposite to capacitive touch resistive touch is mostly resistant to noise as only pressure is also needed to accidentally trigger a touch event. Capacitive touch on the other hand relies on changes in the electric field and can be prone to noise. We want to explore both capacitive and resistive textile touch pads in this thesis.

Contribution

In this thesis we present out embroidered touch pad designs. We explain how to create the touch pads and evaluate their designs. Our touch pads will be used to recognize multiple gesture and used in applications as a proof of concept.

Chapter 2

Related work

This chapter is going to explore related work in the field of interactive textiles. It is split into the work on two dimensional textiles sensors that implement multi-layer solution, work around wearable textile touch sensors as well as embroidered touch sensors.

2.1 Two dimensional textile touch sensors

Textile Based Capacitive Pressure Sensor

Previous work on textile touch pad sensors was focused on multi-layered approaches. Mostly two conductive layers that are separated by a non-conductive layer. These are then aligned manually and require lots of labor to be constructed properly. ”A Textile Based Capacitive Pressure Sensor” by M. Sergio and Canegallo [2002] uses two fabric layers lined with conductive fabric separated by a foam layer between those. Their goal was to create a low cost, flexible and simple pressure sensors. They equated presses with a small surface area resulting in less change in capacitance with light strokes. By using creating a grid with their two separate conductive layers they were able to create a 24 by 16 pixel sensor with 384 intersection points. As a proof of their work they were able to recreate a rough image of a fist touching their sensors by reading out their input.

2.2 Embroidered touch sensors

The Textile Interface Swatchbook

S. Gilliland and Zeagler [2010] created a collection of textile widgets with the use of an embroidery machine. These widgets over different ways to interact with graphical user interfaces. They focused on novel interactions and provided insights on embroidering their designs successfully.

Embroidered EMG sensor

Linz [2007] used embroidery machines to create a small wearable capacitive EMG sensor. They connected their multi-layer sensor to a flexible PCB by embroidering thread onto it as a connection point. They were able to deliver similar results as contact electrodes by reducing the noise of their sensor and were reliably able to distinguish resting periods from exercising ones.

2.3 Wearable textile touch sensors

Inviz: Low-power Personalized Gesture Recognition using Wearable Textile Capacitive Sensor Arrays

Gurashish Singh and Banerjee [2015] used conductive thread to create a robust wearable sensor for disabled people with limited mobility. Relying on imprecise and proximity based gesture detection users are able to control smart homes. The sensor aims to be unobtrusive while being easy to use to put minimal amounts of strain on the users. Initially they worked with capacitive patches but later on went on to use embroidery to automate the creation of their sensors to study the scalability of the production of said sensor.

GestureSleeve

Schneegass and Voit [2016] used a multi-layer approach to create a smart sleeve to augment the forearm with touch functionality. It is envisioned as a touch input device for smart watches to bridge the gap between small touch interfaces and mid-air gestures. The sensor consists of three layers. On top and on the bottom are 32 parallel stripe electrodes and in between is a layer of piezo-resistive fabric. This results in 1024 pressure sensor points to sense taps and gestures. Their prototype out performed regular touch interactions on the smart watch in terms of completion time and they want to further evaluate other application scenarios.

Chapter 3

Fabrication

Fabrication setup

All of our prototypes presented in this theses are designed using their proprietary Bernina Embroidery Software 8 Designer Plus software and created with the use of our Bernina B880 embroidery machine. We used a embroidery needle with a 75 needle size and a straight stitch needle plate. We used a hoop with the dimensions of 26.5 cm by 16.5 cm to fit our designs. Lastly we used a universal drop-shaped embroidery foot, Bernina's embroidery foot #26. Finally when we were tasked with trouble shooting the bernina we referred to their trouble shooting guide[1]as a reference for embroidering

[1]Troubleshooting Embroidery Designs

3.1 Choice of micro controller

Definition: *Relaxation Oscillator*

RELAXATION OSCILLATOR:
A relaxation oscillator is in our context of use a nonlinear oscillator circuit. The circuit consists of a feedback loop using a comparator that repeatedly charges a built in capacitor until it reaches a certain threshold. Then the capacitor is discharged again and the cycle repeats. This results in a waveform of charge and discharge cycles. Typically the amount of these cycles are then measured within a fixed time frame with standard relaxation oscillator method. A increase of capacitance through a touch thus means a decrease of cycles within that time frame.

Starting out with an Arduino Uno

We used an Arduino Uno at first for all of our prototypes. The Arduino Uno uses a software approach to mimic an relaxation oscillator to measure capacitance in processor ticks. The time needed to change the pin state is measured from 0 to 17 ticks. We connect this cell to our micro controller to use it as a capacitive sensor. This approach however is very limited as a not having the actual circuit for an relaxation oscillator on board of our PCB makes for a cruder sensing. This is because the board has to use the digital pins to see if they are low or high and measure in ticks how much time it takes with the current capacitance until they change states. Another problem with this is that lower changes in capacitance are not recognized because the the pin never reads high. Because of this interaction with the insulated lower thread are not recognized. This way we can not use our Arduino Uno for designs with fully insulated lines. But we can use our Arduino for resistive designs with only partly insulated lines.

Using a MSP430 for more precise sensing

Realizing the limits of our Arduino Uno we change our prototyping board to an MSP430G2553 with an on-board relaxation oscillator circuit for capacitive sensing. The MSP430 allows for a more delicate sensing ranging in values from as low as a hundred cycles to multiple thousands to differentiate touches. The biggest limits we encountered with this micro controller is the limited amount of pins. Looking for more pins being able of capacitive sensing we thought of using the SMD version of this micro controller with a cus-

tom PCB to allow for a higher resolution. This would have went beyond the purpose of this thesis and may be subject of future work.

Alternatives

To allow for higher resolutions without getting into custom PCBs and soldering we explored the Tiva C shortly for a higher resolution pin offering potentially 40 digital pins for a 20 by 20 resistive sensor. We did not explore this further as we were more invested in capacitive sensing as it provides a better user experience by recognizing even faint touches. Lastly we used the LilyPad as well an Adafruit MPR121 for more wearable options. The LilyPad uses the same library for capacitive sensing thus offering no benefits other than being more wearable. The Adafruit MPR121 was very promising being designed for capacitive sensing but testing it we ran into problem again with our insulated lines. We could not get the micro controller to recognize touch input on insulated lines and thus discarded that board and focused ultimately on the MSP430G2553.

3.2 Our first Prototype

PITCH:
The pitch of a sensor refers to the spacing between the threads on the rows or columns. E.g. a sensor with 9mm pitch means that the columns and rows are spaced 9mm apart from another

Definition:
Pitch

Creating a 4 by 4 grid

The first step in towards creating our first prototype is creating a design for our sensor in Bernina's proprietary design software. We simply create a 4 by 4 grid by aligning 4 horizontal and 4 vertical threads. On top of the horizontal lines we place zic-zac embroidery patterns to insulate the lower thread. By color coding all those separate pieces of our design we can later embroider them according to their assigned color in the software. This way we first embroider the horizontal lower thread using our conductive thread. Then on top of that we embroider our insulation using the non-conductive thread. Lastly we use our conductive thread again to embroider our vertical lines on top. This proves to be a tricky step as our conductive thread may

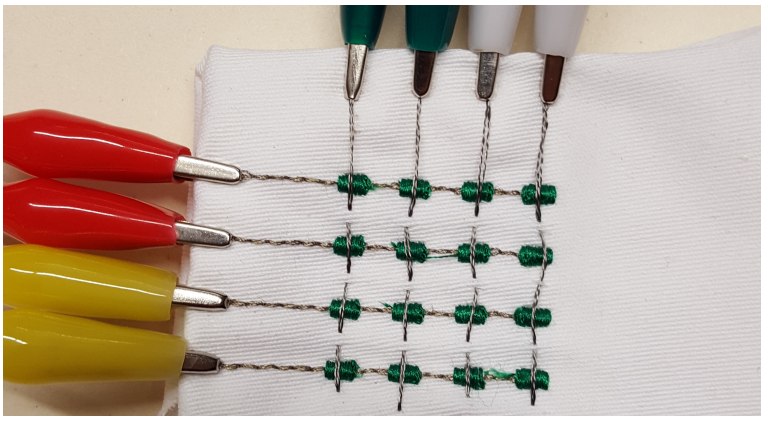

Figure 3.1: Our first prototype featuring a 4 by 4 grid

not pierce through our insulation. That way we would create a short circuit rendering our sensor or at least a parts of our sensor unusable. In our first prototype the last step was done manually as we had to find the right settings first.

Jumping over the insulation

At first we tried to force the embroidery machine to jump from one continuous line to another. This did not work as the thread is automatically cut jumping from on line to another. While we could have followed this approach we realized this would yield a fragile connection between the lines at best as the single connection point could prove to be a breaking point. So we followed a simpler approach in dividing the upper thread into to two sections. The first on is a jumping stitch that stitches into the fabric every pitch distance. So if we have 9mm pitch our jumping stitch also jumps 9mm. We adjust our jumping stitch to always stitch into the center of our pitch and then jump over the insulation until it has jumped over every insulation. From there our stitching pattern changes to a regular stitch that we embroider outwards to have space for our alligator clips to connect our board with the fabric.

Choice of threads

There are a multitude of threads available for use in smart textiles. They can be made up completely made up from conductive material such as stainless steel. The result is less prone to wear but often more hairy and harder to control in a embroidery machine as the machine has problems managing the frays. Other approaches use a nylon core for

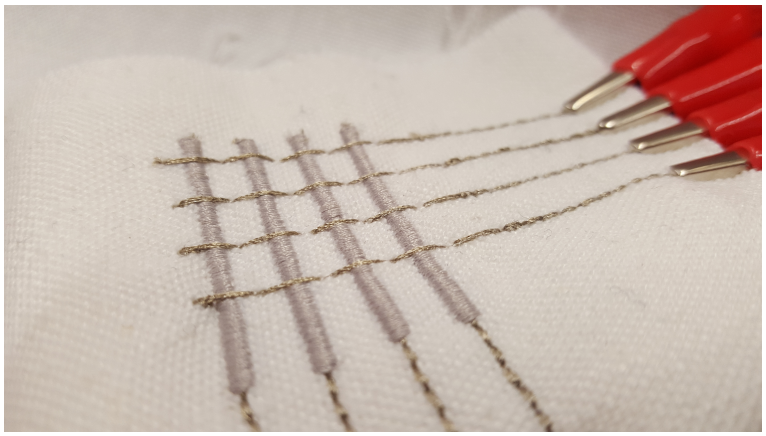

Figure 3.2: Side view of our jumping pattern not piercing our insulation

example and plate it with conductive material such as silver. Because it consists mostly of nylon it also behaves similarly to nylon and can be used just like normal thread inside a embroidery machine. This approach comes with the down side of being prone to wear as once the silver plating is worn of the thread is no longer conductive opposed to a fully stainless steel thread. We decided to use a nylon based conductive thread as the reliability proves to be more important than the longevity of our sensor as our embroidery machine was not able to produce reliable textile sensors for us to use with other threads.

3.3 Finishing a prototype

Cutting our sensor

In order to improve the output of working sensors we added additional steps creating a sensor. After our horizontal lower thread has been embroidered we remove the hoop from our embroidery machine. The embroidery machine waits for confirmation to embroider the next thread allowing us to later add our hoop back without missing a step. We then remove every stray thread of conductive thread under our thread and above it without removing the fabric from the hoop. When possible threads should be pulled out lightly without force.

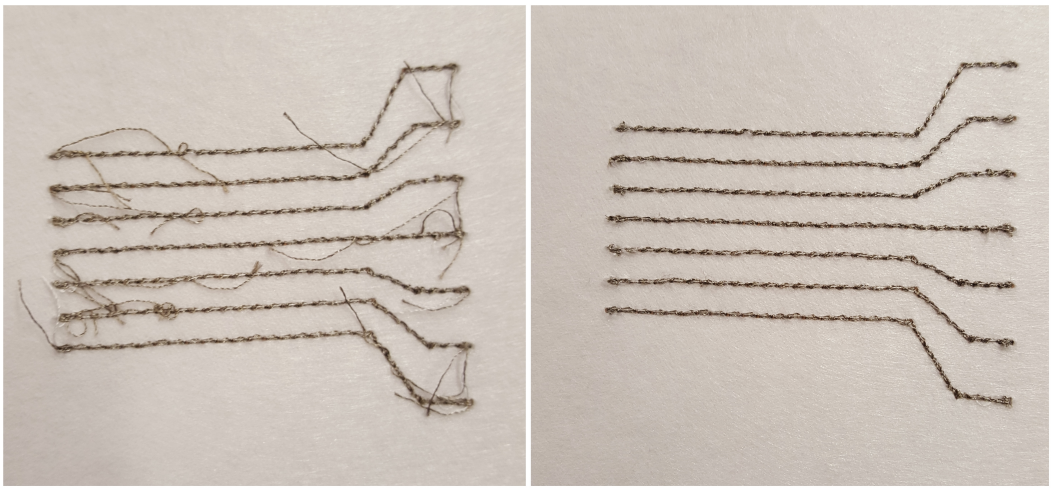

Figure 3.3: Left: Thread under our sensor before cutting Right: After, ready for insulation

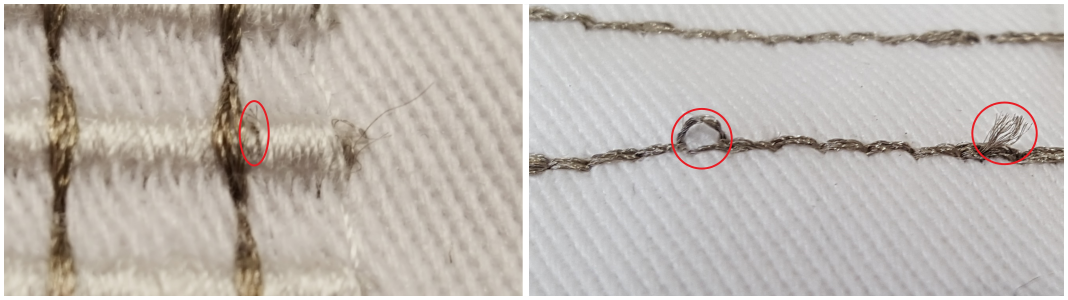

Figure 3.4: On the left broken insulation caused by loose thread on the right

Thread loops

Additionally sometimes while embroidering the embroidery machine creates little loops because the bobbin thread did not pull tightly onto the thread. Because we use a triple stitch these, they may be cut when they are to excessive to discard any chance of this loop later on piercing our insulation allowing for short circuits on our sensor. Sometimes these loops can be evened out by pulling the thread next to the loop to pull the loop somewhat back into the fabric allowing the insulation later on to correctly jump over the thread and insulate it.

Continuity testing

After our grid is done we have to ensure there are no short circuits. To do this we use a multimeter. We test continuity over the whole grid connecting the multimeter probes to

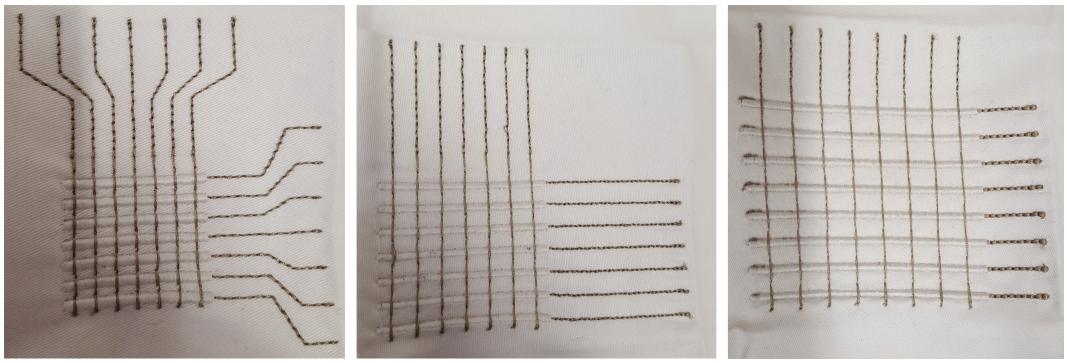

Figure 3.5: Left to right: capacitive design with 5mm, 7mm and 9mm pitch

each row and column in every possible combination. Once we detect a connection we check for stray threads and pull it out or cut the thread and continue testing. This is usually not the case though as our sensor should come out without short circuit if the threads were cut accordingly and the embroidery machine worked without error.

Connecting the sensor

We are using alligator clips with pigtails to connect our grid with our micro controller. Previous attempts with wire piercing the thread did not create stable enough connections and separated easily. The alligator clips help creating a sturdy connection with low resistance through a big contact area for our thread to connect to.

3.4 Micro controller programming in Energia

CAPTOUCH:
We make use of the CapTouch library provided by Energia[2]to support the Capacitive Touch Booster Pack made by Texas Instruments for the MSP430 Launchpads. This library provides a high level abstraction of the registers that are used to read out the current capacitance in cycles. Additionally it manages a baseline to adjust for changing levels in capacitance. It compares this baseline to the currently measured capacitance and checking whether a threshold is passed. If that is the case a touch is detected.

Definition: *CapTouch*

Low level and high level programming

In terms of software there are two components. The first component is the code running on the microcontroller getting the input and the second one is the code running on the computer using said input. The first component is our Energia code running on the MSP430G2553. For each horizontal and vertical threads we have a pin connected. These pins are each associated with a CapTouch instance. The pins are split. The first part is used for the vertical threads counting from the left to right and the second part is used for the horizontal threads counting from top to bottom. These are defined under `gridy` and `gridx` respectively. Each pin is internally connected to the relaxation oscillator and it's capacitance is measured in cycles through that oscillator. From that the base line is derived. If a pin or the thread connected to that pin is touched there is a change in capacitance. If that change oversteps a certain threshold the touch is recognized. The difficulty there in lies to be as lax as possible to recognize even the lightest touch but strict enough to deny false positives. For each pin we sent a either a '0' if it is untouched or a '1' if it is touched over the serial port. After all pins are sent a line break determines the end of one input set.

Filtering the touches to increase reliability

While our setup does support multiple inputs from various pins at once we have noticed improved reliability with restricting the pins we actually read out. Currently we implement three different modes of our Energia code to filter the input we receive on our board. Our approach is quite simple the strongest filter only gives out the intersection with the highest values currently and discards all other inputs. Of course our basic threshold has to be surpassed anyways. This is the most robust approach allowing interaction as expected under almost all circumstances as an touch provides always higher values than noise disturbing the sensor. With this approach however we can't support multi-touch or interpolation to increase the accuracy of our sensor.

Recognizing only one touch

To improve upon this while not letting our sensor be totally unreliable when noise disturbs our sensing we implement a hybrid approach that still seeks out the intersection with the highest values but instead of only reading that

[2] http://energia.nu/

input also reading out input from the rows and columns around it if the threshold is surpassed on those as well. This allows us to implement interpolation to increase the precision of our sensor while decreasing the robustness of our sensor. Lastly we also have a simple multi-touch mode that recognizes each pin that surpasses our given threshold allowing ideal interaction if noise does not disturb our sensing.

Adjusting baseline as a reference point

To further improve the performance of our sensor we make us of the adjusting baseline of the CapTouch library. In a perfect scenario our touch detection would work as follows. If our touch measurement jumps to such a an extend that it exceeds our threshold we recognize a touch and we only release that touch if we drop below our threshold adjusting. To do that we have to have a baseline to compare to as we can not rely on previous measurements alone.

Adjusting the baseline

At the very beginning of our code we just set our starting base according to a rough average of our last measurements. Assuming there are no touches occurring at that time we have a representation of our measurements while there is no touch occurring. We use this baseline for our touches from now on. Every time a measurement exceeds our threshold compared to our baseline we recognize a touch and do not change our baseline. If however no touch occurred when going through a certain pin we will level the baseline for that pin to approach our average measurements after the last recognized touch. We do this because no touch means we should be a neutral state not slowly building up towards a touch resulting in a false positive. Neither should our baseline be decreasing resulting in unregistered touches. So in case of our delta between the baseline and our current measurement being positive we will add our baseline and that measurement and divide them by two slowly bringing our delta over the course of our next measurements towards zero. If our delta is positive we do not decrease it so rapidly at first because we might adjust our baseline too much in case of the capacitance rising when a touch occurs. That way a touch that would barely make our threshold would fall short and therefore would not be recognized. This is the reason why we only slowly decrease our baseline by one every time we check for touches when our delta is above

zero.

3.4.1 Debugging

Checking for proper connections

Once our code was up and running we had to check if our sensor is performing as expected. For this we implemented a debug mode transmitting our current baseline as well as the delta between our current measurement and baseline. We simply check at first that touching the exposed thread on each row and column without touching the alligator clips provides deltas in the range of 2000 cycles at least. If that is not the case we adjust the alligator clips and press them down firmly to connect them tightly to the fabric. When all our rows and columns are connected properly we go on to set the thresholds.

Setting thresholds

At first we set the threshold of all pins connected to vertical threads to 800 cycles and all pins connected to horizontal threads to 150 cycles. Then we inspect our serial bus to ensure that when we lightly touch a intersection point both the horizontal and vertical pin are being touched indicated by a '1' being transmitted on their position. If that is not the case even though the sensor is touched we must lower the threshold accordingly. We check delta when we touch the thread to see what threshold would be appropriate. The threshold should be the lowest delta recorded while touching the thread to ensure even a light touch is detected. If that is not possible without noise and/or the pins next to the one misbehaving then we should go back to ensure proper connections and maybe even create a new sensor.

Pins getting stuck

Additionally pins may be indicate that they are being touched without that being the case. This can be remedied by raising the threshold slightly over the delta. This then allows the baseline adjusting to kick in leveling out our baseline to adjust for the capacitance on that pin.

Limiting the serial transmission

After having completed our debugging process making our sensor work smoothly we disable the debug flag in order to only transmit the information that is necessary to run our processing code. This is either only transmitting the delta levels on our pins separated by tabs or a single bit string

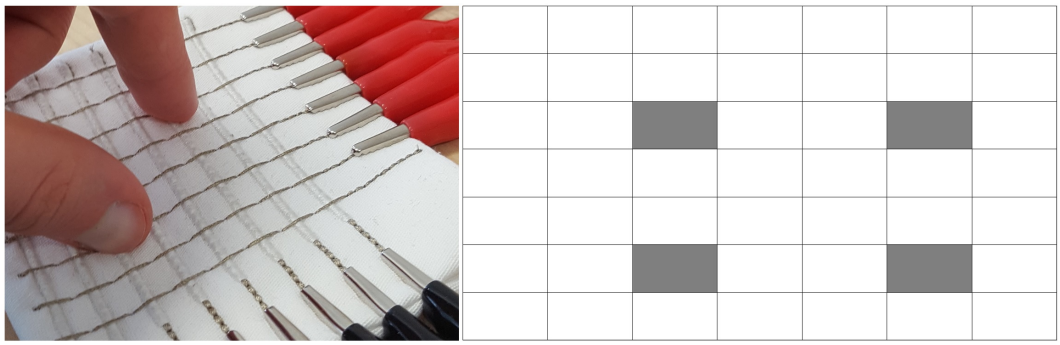

Figure 3.6: Two finger input triggers 4 touches as our input does not allow to discern which are actually occurring

containing which pin are touched. We do this to limit the serial transmission which slows down our sensing noticeably because our interrupt time increases as we have to wait until one line is being transmitted and processed.

3.5 Applications in Processing

Processing Code running on our PC

The second component is our code running on our PC using Processing. Here we receive the input stream and parse it into a 32-bit integer variable `input` whenever a serial event occurs. The application then determines using bit shifts on our integer whether a pin was pressed or not. We go through the first seven bits to determine which horizontal thread was touched and then through the next seven bits after those for the vertical threads. This way we see all possible intersections given our input stream.

Multi-Touch issues

Though this technically allows multi-touch the input is not processed correctly as one might expect. All multiple presses on a diagonal line trigger always the amount of presses squared. That happens because every possible intersection point according to the pin assignment is triggered regardless of the actual touches occurring. This is resolvable by timing the input and deriving the correct intersection points based on the sequence of pins being set. Because it is nearly impossible to have multiple touches occurring within one interrupt this should be a plausible solution to that error.

3.5.1 Interpolation

Definition: *Interpolation*

INTERPOLATION:
In a mathematical sense interpolation means finding a continuous function for given discrete points. In context of this paper this means using given sensor input and concluding the actual position of the finger on our touch pad. Thus interpolating the actual position of the finger from out discrete input data.

Increasing resolution

In order to increase our resolution on our limited 7 by 7 grid with only 49 intersection points we apply a simple form of interpolation. This way we increase our software grid to a size of 13 by 13 with 169 different touch points. We do this by differentiating touches on a intersection point that only trigger one pin on each axis opposed to touches that are off the intersection point between two threads triggering both respective pins.

Calculating the center

Previously going through each half of our pins in our variable `input` we stored each touched pin for different touch event. Now we add the positions of every pin that is touched up and increase our counter by one for each. We then divide the sum of positions by our counter calculating the center position of all pins. We then multiply that position with two to adjust for our rectangle array indices. With our previous indices ranging from 0 to 6 they now range from 0 to 12. To visualize which point is currently being touched we color the rectangle related to that index grey.

3.5.2 Gesture

Storing time

To recognize gestures we need to remember in which sequence our touches occurred. To do that we simple associate a time whenever a touch occurs and we therefore color a rectangle grey. We store this time in our two dimensional array `hist` with the same indices as our `rect` array. If no touch has occurred on given indices the value on that entry is zero. With this we can iterate over `hist` to determine if 3 touches occurred in sequence on a row or column.

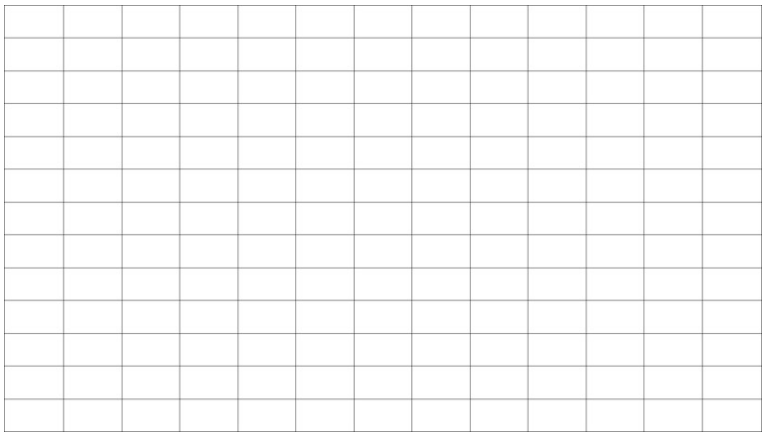

Figure 3.7: Our software 13 by 13 grid created by interpolating pins

Processing our history

At first we go through each row of our array checking if the order of values is increasing. We disregard entries with the value zero doing this and stop after having found 3 entries with increasing values. This would indicate that the vertical threads on our grid have been touched from left to right indicating a swipe to the right. Then in a similar fashion we check if the order of values is decreasing while disregarding zeros. This would indicate a left swipe on our grid.

Transposing our array

Having checked for all possible left and right swipes on each line we transpose our array turning our columns to rows and we repeat our process. We check for increasing and decreasing orders of values indicating down or up swipes on a given column. If we did find a swipe we visualize it by coloring that row or column according to the swipe detected as seen in the figure 3.8

Resetting the history

In order to continually recognize gestures we need to reset our history from time to time to correctly access our history and deter false positives as well as unrecognized swipes. We do this after 500ms when there has been no input on our grid.

Implementing snake

Having implemented our simple gestures we use them to control our implementation of snake. In order to control the snake we have to swipe in the direction we want it to go. Every time a swipe is registered we reset our history to allow the user to leave his finger on the grid.

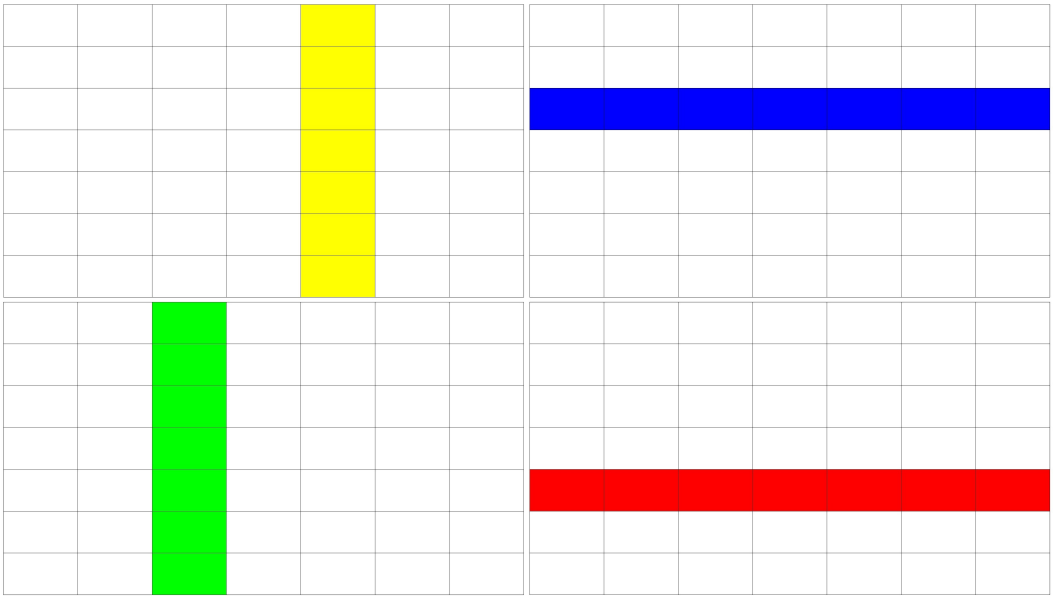

Figure 3.8: Simple swipes, green is down, yellow up, blue left, red right

Figure 3.9: Simple snake game controlled by our touch sensor's gestures

3.5.3 Pressure Sensing

RGB COLOR MODEL:
The RGB color model is additive color model that color describes colors by separating them into their proportion of red, green and blue light on a scale from 0 to 255. This way every color display with this model is described using these three values. The values (0,0,0) refer to black while the values (255,255,255) represent white.

Definition: *RGB color model*

Relating touch area to pressure

By increasing the touched area on our threads we increase the change in capacitance and thus our measured cycles. M. Sergio and Canegallo [2002] has shown that you can use that increased change in capacitance to correlate the pressure of the touch. As we touch the grid with more pressure we also increase the touch area of our finger on our grid increasing the change in capacitance. Thus an bigger change in capacitance correlates to more pressure.

Working with delta

In order to get the necessary information for our pressure sensor we change our micro controller code to transmit the delta between our current measurement and our baseline of each pin. We then found the values for a strong touch and used them as the upper boundary. We used our already set thresholds as the lower boundary. Each delta transmitted is then checked if it passes our lower boundary and if it does it is then divided by our upper boundary. This is then the percentage of touch strength. We do this differently for our horizontal and vertical threads as the insulated threads have smaller changes in capacitance. This means as we have different thresholds for our threads we also have different upper boundaries for our pressure sensing.

Visualizing our pressure

To visualize the pressure on a given intersection point we color it according to the pressure on that point. With increasing pressure the percentage of our delta in relation to our upper boundary increases. We map this factor in a linear fashion on the RGB color model. We do this by multiplying our factor by 127.5. Doing this on both pins representing our horizontal and vertical threads we can add this up to a maximum value of 255 for an intersection point. Using the RGB color scheme we start out with a value of 255 for red, green and blue. We subtract our calculated RGB

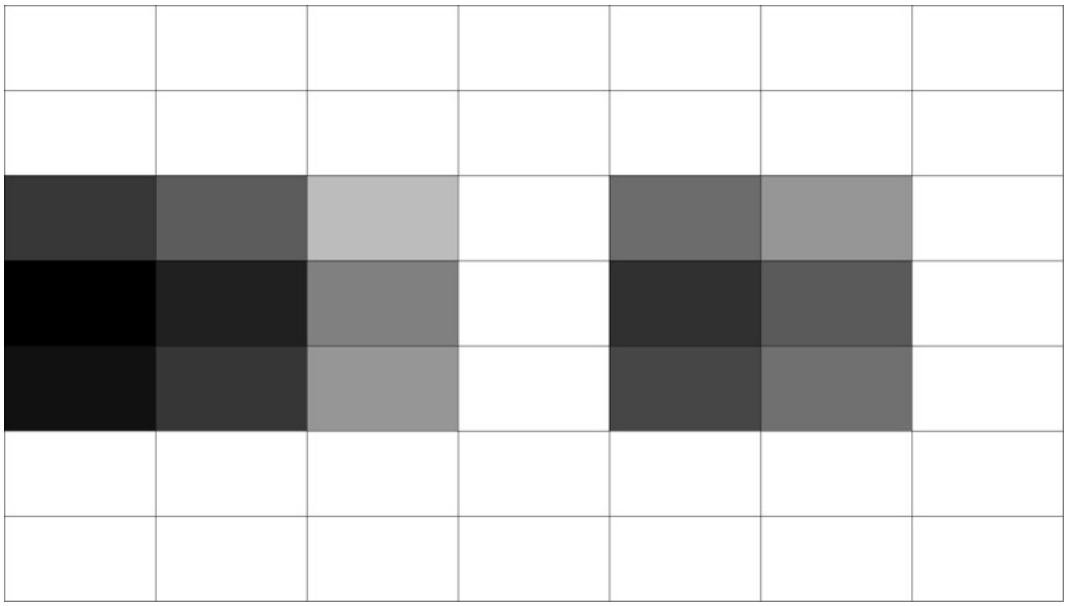

Figure 3.10: A visualization of the pressure levels on each intersection point of our grid. Darker means more pressure

that we based on our delta level from that. This means as our factor increases the intersection point gets darker as seen in figure 3.10.

3.6 Resistive Sensing

Adjusting the sensor

The idea behind our resistive sensor is that we can use our fingers to bridge the connection at intersection points. This way we have to change our design as we can not full insulate the horizontal threads. For our resistive design we only partly insulate the horizontal threads at intersection points in order to avoid short circuits while still allowing our fingers to connect the vertical threads with the horizontal ones.

Recognizing a touch

Typically we would go through our grid setting one pin as an input and the other pins as output and high and check if our input pin changes states. This proves to be impossible with the high resistance of our fingers and our thread. So in order to recognize a touch we used a code snippet from Arduino Playground[3]. They set out to measure the capacity

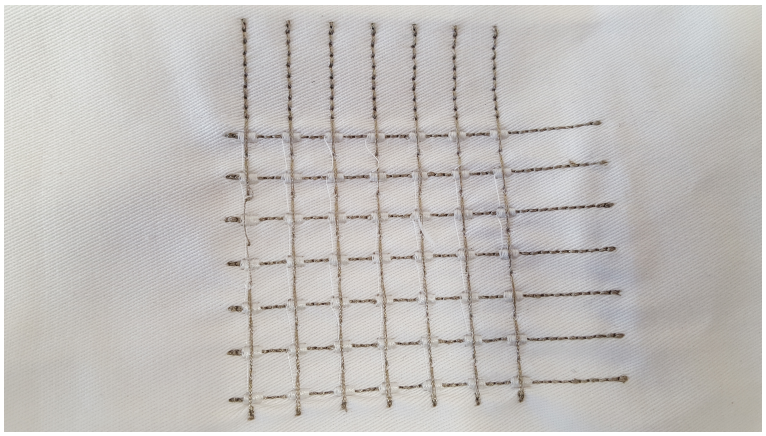

Figure 3.11: Resistive design with 9mm pitch

of a pin using the internal pull-up. They resistance of our finger would slow down the time the pin takes to pull up after discharging that pin. This time is measured in hardware ticks by checking every hardware tick 17 times if the internal pull-up of the pin is now on. After that the pin is set to output and low discharging the pin again. While it is called a capacitive sensor it does not contain a capacitor making it resistant to noise. Every time we connect the threads with our fingers at an intersection it takes one or two hardware ticks until the internal pull-up is on on these pins.

Transmitting a touch

So every time we measure a value higher than zero we transmit a '1' instead of a '0' for that pin similar to our capacitive setup. We process this exactly as we do with our bit strings for capacitive sensing to determine which intersection point is being touched.

[3] Arduino Playground Snippet

Chapter 4

Evaluation

In this chapter we will evaluate the success rate of our prototypes. We will be using our 7 by 7 prototypes using capcitive sensing with differently sized pitches.

Testing our prototypes

4.1 Evaluation Setup

Maurin Donneaud and Strohmeier [2017] proposed in their paper ”Designing a MultiTouch eTextile for Music Performances” a standardized evaluation in order to gauge the accuracy of textile touch interfaces. They measured the accuracy of their prototype by comparing it to a ground truth. In this case the ground truth is the physical input on the prototype. This is then compared with the input the prototype receives and maps as a physical representation of the received input.
To do this we marked randomly 10 spots on a blank piece of paper with the dimensions of our respective prototype. We did this to decrease any possible bias in choosing touch points. We then transferred these spots onto our prototypes and measured the distance on the x-axis and y-axis from the top left of our grid. The position of our spots is our ground truth we use as a measurement for accuracy.

Comparing to ground truth

Then we map the input our micro controller gets. Each

Mapping our input

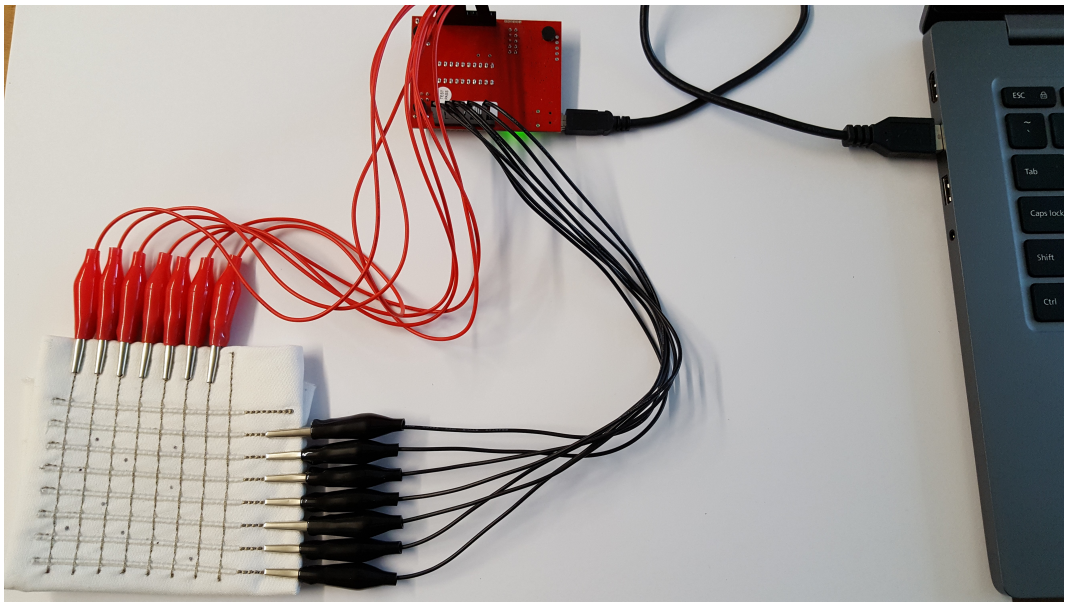

Figure 4.1: Our evaluation setup. The black marks indicate the desired touch points

pin resembles are row or a column on our grid and thus a x-coordinate or a y-coordinate on it. If multiple pins get touched we take the center of them for up to 3 pins on a row or column. This way we should be able to accurately map touches in jumps of half of our pitch size. On our grid with a 9mm pitch this means we can map with a 4.5mm interval. So in theory the worst we are off on our 9mm pitch should be 2.25mm.

Differences in pitch sizes

Because our prototypes are not perfectly square these dimensions are off for our horizontal lines which are used to map our y-coordinates. Their interval still remains half of their actual pitch size with the highest expected offset being half of the interval they can map. Having considered all of this we write the mappings of our input into a text file marking each touch with the current point we want to touch as well as the program time to avoid any confusion. If we have touched a point 10 times we press the next number starting from 1 up to 10 to write the log for our next point. After touching our 10th point for 10 times we press enter and go on to evaluate our resulting log file.

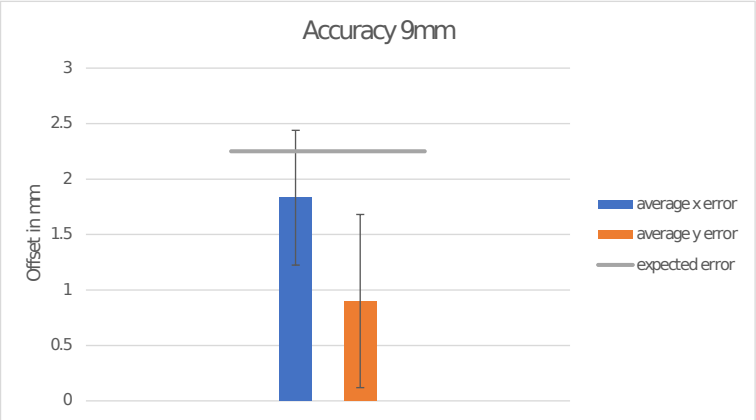

Figure 4.2: Accuracy on our 9mm pitch (54mm x 52mm)

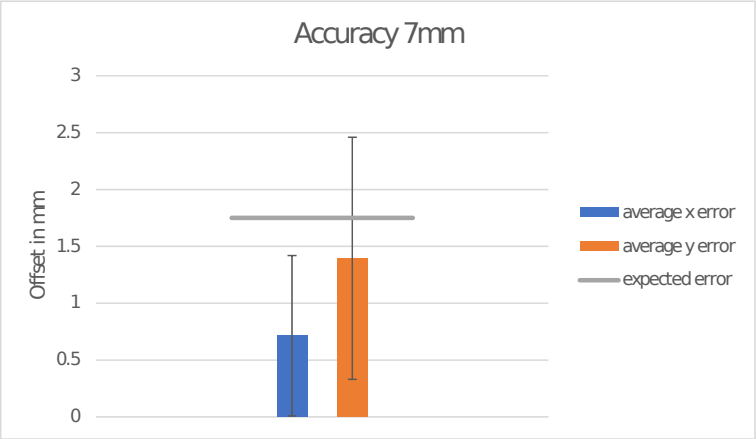

Figure 4.3: Accuracy on our 7mm pitch (42mm x 40mm)

4.2 Evaluation Results

Visualizing our data

Taking the data from our log files we subtract the touch points we measured from the one we mapped and we take the absolute of that. This is our error for each touch. We average this for our x- and y-coordinates. We also take the standard deviation of our error values to visualize the possible variation of our results. Lastly we add a horizontal line to show the highest expected error on our given pitch size.

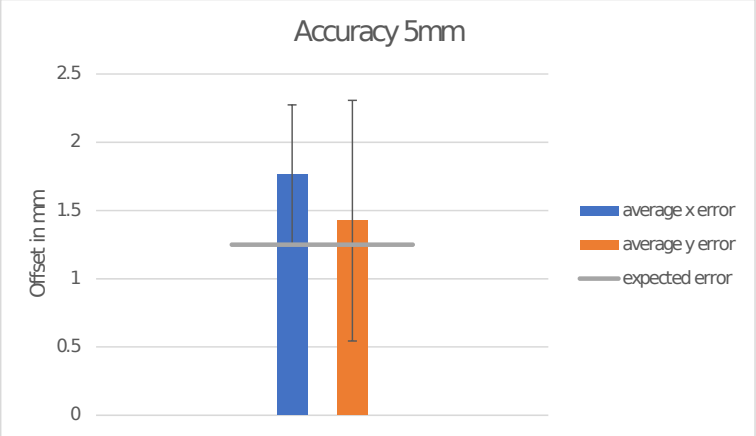

Figure 4.4: Accuracy on our 5mm pitch (35mm x 34mm)

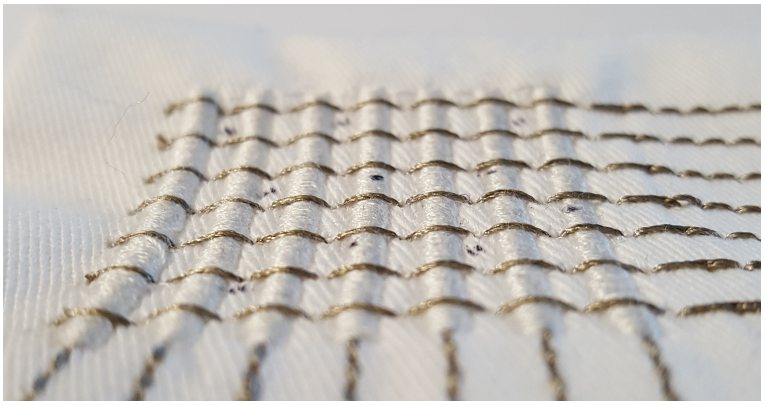

Figure 4.5: Side view of the insulation on a 5mm pitch taking up most of the prototype

Evaluating our results

The results for the prototypes with a pitch size of 9mm or 7mm fall into our expectations as our expected error was not exceeded. The standard deviation is quite high as each touch we enter is different from the one before just by human error. Additionally every time a touch is off a deviation of up to half the pitch size is possible allowing even few deviations in measurement over our 100 touches to create a high standard deviation.

5mm pitch under performing

More surprisingly though was our prototype with a 5mm pitch exceeding our highest expected error on both x- and

y-coordinates. This is due to our prototype under performing compared to our other prototypes resulting in inaccurate mappings. It is also notable that on this prototype the insulation takes over most of the grid making it very hard to touch the fabric in between the insulation possibly increasing the error even further.

Conclusion

Looking at these results we can conclude that our prototype with a 7mm pitch is best at mapping a touch accurately with around 1.75mm error due to covering a smaller area than our 9mm pitch but also because it works more reliably than our prototype with a 5mm pitch.

Chapter 5

Summary and future work

5.1 Summary and contributions

We created a easy to recreate two dimensional textile touch pad sensor using an embroidery machine and conductive yarn. Our sensor is capable of recognizing touch inputs and simple gestures reliably in real time with little delay. On top of that we can sense pressure with our capacitive sensor in a rudimentary manner. Because we embroider our design on one textile layer with multiple layers of thread we refined our design to reduce possible short circuits that would destroy our sensor. We encountered troubles with our sensing in resistive as well as capacitive design. In the resistive design we had problems with the high resistance of our fingers. The capacitive design naturally encountered troubles with noise especially when coming into contact with a users.
Our design is scalable in size and resolution to accommodate different needs. Because we work on a single layer creating our design with an embroidery machine we can easily bring our design to existing clothing or other fabric. We describe how to create these sensors and how to implement them using our code. Furthermore we evaluated our capacitive sensor showing it can map touches up to one or

two millimeters error accurately.

5.2 Future work

The most important step in future work is implementing our design into existing fabric with the most challenging application being clothing. This will need improved robustness for our capacitive design and higher responsiveness for our resistive design to work reliably in such a circumstance.
This means implementing shielding techniques for our capacitive design to shield the sensor from possible noise of the human body especially in motion. More over the micro controller as well as the wiring has to be adjusted to be wearable and must be shielded as well. Using our sensor in clothing also means that we have to greatly improve it's resistance to wear as our sensors are susceptible to wear. Additionally we have to adjust our design with small pitch sizes in order to let the sensor remain flexible instead of becoming increasingly stiff with the relatively increasing amount of insulation
Furthermore with increased stability of our design multitouch could be implemented allowing complex gestures allowing for many different use cases.

Appendix A

Chapter 6

Source Code and Bernina files[a]

[a] https://hci.rwth-aachen.de/embroideredTouchpads

Bibliography

Ryan Robucci Chintan Patel Gurashish Singh, Alexander Nelson and Nilanjan Banerjee. Inviz : Low-power personalized gesture recognition using wearable textile capacitive sensor arrays. In *In Pervasive Computing and Communications (PerCom), IEEE International Conference*, 2015.

Shiho Fukuhara M. Emre Karagozler Carsten Schwesig Ivan Poupyrev, Nan-Wei Gong and Karen Robinson. Project jacquard: Manufacturing digital textiles at scale. In *Proceedings of the 34th Annual ACM Conference on Human Factors in Computing Systems, ACM*, 2016.

L.; Langereis G. Linz, T.; Gourmelon. Contactless emg sensors embroidered onto textile. In *Proceedings of the 4th International Workshop on Wearable and Implantable Body Sensor Networks, Aachen, Germany*, volume 13, pages 29–34, 2007.

M. Tartagni R. Guerrieri M. Sergio, N. Manaresi and R. Canegallo. A textile based capacitive pressure sensor. In *Proc. IEEE Sensors, vol. 2, no. 12–14, pp. 1625–1630*, 2002.

Cedric Honnet Maurin Donneaud and Paul Strohmeier. Designing a multitouch etextile for music performances. In *Proceedings of the 17th International Conference on New Interfaces for Musical Expression*, 2017.

T. Starner S. Gilliland, N. Komor and C. Zeagler. The textile interface swatchbook: Creating graphical user interface-like widgets with conductive embroidery. In *Proc. ISWC '10*, pages 1–8, 2010.

Stefan Schneegass and Alexandra Voit. Gesturesleeve: using touch sensitive fabrics for gestural input on the fore-

arm for controlling smartwatches. In *ISWC*, pages 108–115, 2016.

Index

Typeset April 23, 2018